VZ-LINK NEWSLETTER

<< AUCKLAND VZ 300/200 USERS CLUB >>

ISSUE # 16 JANUARY 1988

I hope you all had a really super Christmas and Also wish you all the best for this New Year.

The workshop went really well,even if only 4 members did attend. The reason so few attended was my fault because I picked a time so close to Christmas and also left it so late sending out notice's.
Anyway,Thank's to those that did make the effort to attend and I hope you got something out of it.
Thank's also to Peter Patterson for the loan of his laserlink software for us to look at.
A review of one of these is in this issue.

The next workshop will be held on 27Th February 1988 ,so set this day aside and come along.
I plan to hire the local school hall this time,so it will be better organised.
I will included more details next issue along with a slip for you to full in,If you can attend.

This issue see,s the start of the assemby item as promised also the second part of the snoopy calender along with the changes to be made for printer/plotter use.
We also have part 5 of Robert Quinns UYVZ item .I hope you are finding this series worth while and have got some help from it,There is only one more part to go.

I have had to hold over the Speech Synthesiser notes again,they will be in the next issue,but if you need them urgently just ring me.
(sorry David but space is a problem)
I have also got it off the grape vine ,that someone from Dick Smith NZ will be contacting me,re better support for the VZ wouldn't it be nice?.
I will believe it when I see it.

Anyway I hope there is something in this issue for everyone.

Please remember that any local item,s for use in the newsletter would be really great,so send them in folk,s.

Once again,until next issue--ENJOY

VSWR CALCULATOR BY K.WRIGHT

```
1 REM VSWR CALCULATOR, K.WRIGHT,10/12/86
10 CLS:INPUT"DO YOU WANT HARD COPY";A$
20 CLS:COLOR2,1
30 PRINT@12,"V S W R
40 PRINT:INPUT"FORWARD POWER";PF
50 INPUT"REFLECTED POWER";PR
60 PRINT:PRINT"IF MEASURED AT ANTENNA, THEN :-
70 PRINTTAB(10);"VSWR =";(1+SQR(PR/PF))/(1-SQR(PR/PF))
80 PRINT:PRINT"IF AT TX.& FEEDER LOSSES=3DB:-
90 PF=PF/2:PR=2*PR
100 IFPR>=PF,PRINT"  IMPOSSIBLE TO BELIEVE ! !":GOTO115
110 PRINTTAB(10);"VSWR =";(1+SQR(PR/PF))/(1-SQR(PR/PF))
115 IFLEFT$(A$,1)="Y",COPY
120 PRINT@480,"ANOTHER (Y\N) ?";
130 A$=INKEY$:A$=INKEY$:IFA$="",130
140 IFA$="Y",RUNELSEIFA$<>"N",SOUND6,3;3,6:GOTO120
150 CLS:PRINT@265,"GOODBYE":PRINT:PRINT:PRINT:PRINT
160 RUN"MENU"
```

REVIEW

REVIEW OF LASERLINKS 'EXTENDED BASIC' SOFTWARE

By Pete Patterson

This program gives the VZ an extra 66 BASIC commands,consisting of extra DOS commands,variables,graphics, sound routines and general programming utilities. To enable saving of RAM memory,the routines are divided into segments,and by using the ONLY and NOT statements,the user can use only the routines he requires.

The following list gives most of the extra commands within their sections.

' VARIABLES' CINT,CSNG,CDBL,DEFINT,DEFSNG, DEFDBL,DEFSTR.
' PROGRAMMING AIDS' AUTO,DELETE, MOVETO,RENUM,FIND<string>,TRON ,TROFF,COMPRESS,OLD.
' GRAPHICS' PLOT,FILL,DRAW,ERASE,XDRAW, CIRCLE,BACKGR.
'SOUND' BEEP<pitch>,<duration>, NOISE<intensity>,<duration>, <volume>.
'FORMATTED INPUT' INPUT!,INPUT@,INPUT USING,STR INPUT,GET.
'ERROR TRAPPING' ONERR GOTO,RESUME,RESUME NEXT,ERL,ERR .
'DISK COMMANDS' UNERA,XDIR,LDIR,XDIR%,LDIR%, ERALL,MERGE,LENGTH,TYPE$,FILE$, NEWDISK.
'MACHINE-LANGUAGE SUPPORT' WPEEK,WPOKE,CALL,DEFUSER,VARPTR .
'OTHER COMMANDS' RANDOMISE,ON GOTO,REP#,FREE, SFREE,FIX,SPEED,CSR,RESTORE< line>,

Extended Basic also allows you to create your own functions with the DEF statement.It has extra String handling capabilities.eg.INSTR(<string1> ,<string2>).This function searches through string1 for the first occurrence of string2.

I was really impressed with the sound commands BEEP and NOISE.The little piezo speaker really does itself proud.With these commands you can get some great sound effects.

Overall a great package that really extends the VZEDS BASIC.A must for the serious VZ user.

WRITE PROTECT OVERRIDE SWITCH :- By Joe Leon

This is to my knowledge the first hardware modification for the VZ Disk Drive. As most of you may be aware it is possible to use the reverse side of your Disks for program storage simply by cutting a matching notch on the opposite side of the Disk. Commercial Notchers are available for that purpose or you can use scissors if care is taken. And now to the circuit.

Only the circuit within the box is needed. The rest is part of the disk drive circuitry and is shown so we can understand what is happening. When a notched disk is inserted in the drive a small light shines through the notch on the disk and illuminates the photo transistor (STR1). This has the effect of grounding pin 13 of U3. When that happens the disk can be written to. If you have a Write Protect Tag covering the notch then a 'DISK WRITE PROTECTED' message apears if you try to write to the disk.

It follows then that to overcome the write protect function we must ground pin 13 of U3. The easiest way to ground pin 13 of U3 is to connect a wire to the 4K7 resistor and the other end to the switch as shown in the diagram. I used a Flashing Red Led as a warning for which no resistor is required if used with 5 Volts.

If you have trouble getting a flashing red led from D. Smiths try TANDY'S. This is a simple project that should'nt present any problems. Only the top cover of the disk drive need be removed and just follow the diagrams. Once installed the switch will allow you to write to the flip side of disks without notching or over a write protect tag on the front side. Exercise care when using the WRITE PROTECT OVERRIDE SWITCH as your disks are no longer WRITE PROTECTED.

While you have the cover off there are two more modifications you can try. When the VZ is accessing the drive you only get a dull red glow from the motor on led. It can be easily replaced by a super bright 5mm LED. The easiest way to do it without removing the circuit board is to cut the pins of the old led as close to the body of the led as you can. Then place the super bright led over the old pins and push it towards the front till it nearly touches the rectangular lens and solder in place. Use caution and a heatsink when soldering the led as they are heat sensitive. Make sure the led goes in the right way. The negative side usually has a flat side on the body. See circuit diagram.

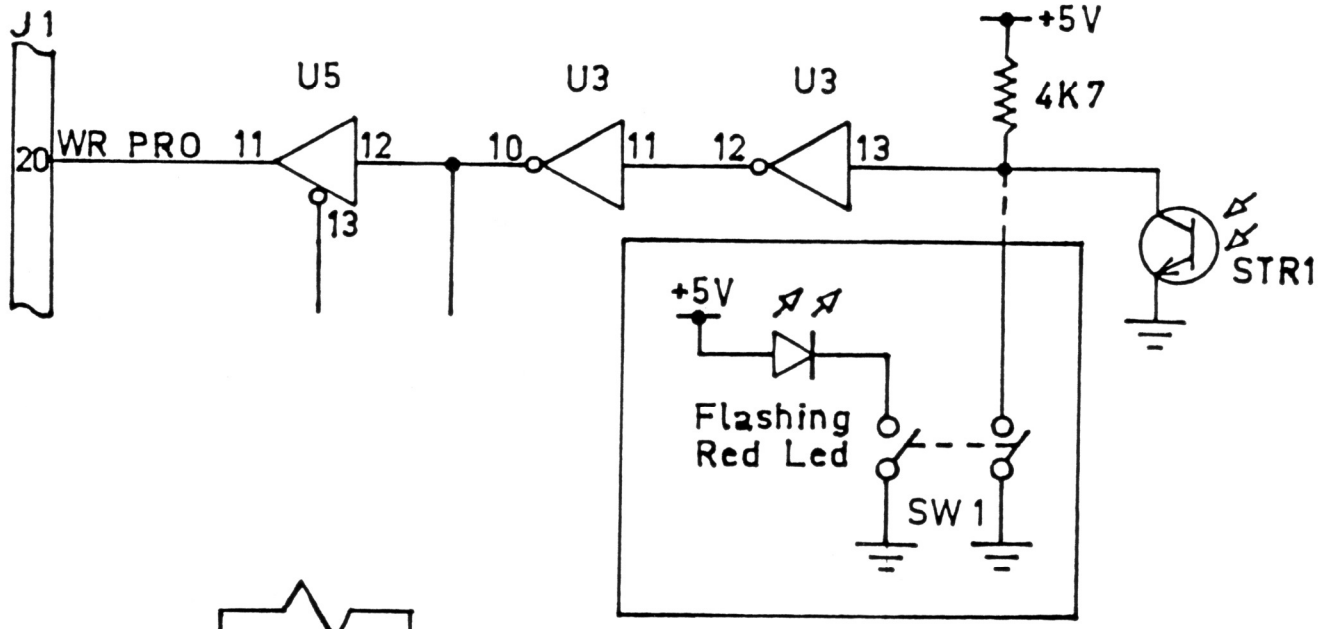
POWER ON/OFF SWITCH :-

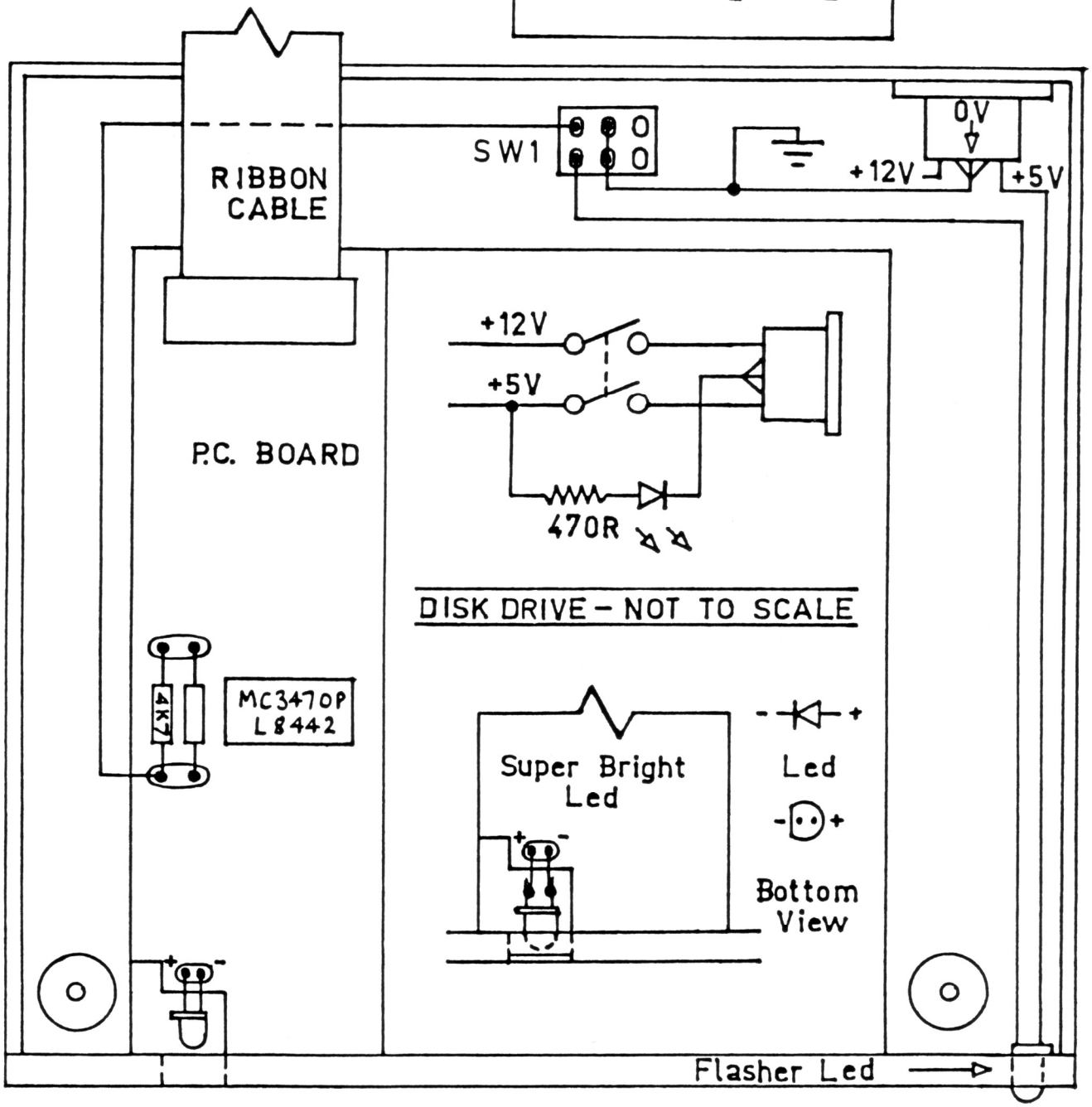
Again this is a simple project. Just a switch and a led with dropping resistor is required. Or you can dispense with the switch and just use the led with its resistor. Again it's up to you where the components are mounted.

If your drive is still under warranty then any modifications will void it. Dont forget the responsibility is yours for any mishaps. If you doubt your ability to carry out the modifications then dont.

TECHNICAL REPORT CONT. - - - -

WRITE PROTECT OVERRIDE SWITCH :- By Joe Leon





SNOOFY CALENDAR BY D.BOYCE - -

```
840 LPRINT
845 '----5----5---X#####X#####X#####X#####X#####X#####X#####X##X
850 LPRINTTAB(30);"              XXXX"
860 LPRINTTAB(30);"             X    XX"
870 LPRINTTAB(30);"            X  ***  X                XXXXX"
880 LPRINTTAB(30);"           X  *****   X            XX     XX"
890 LPRINTTAB(30);"       XXXXX *******  XXX      XXXX         ";
892 LPRINT"XX"
900 LPRINTTAB(30);"    XX     X ******   XXXXXXXXX             ";
902 LPRINT"  XX XX"
920 LPRINTTAB(30);"   XX        X ***    X                     ";
922 LPRINT"   X** X"
940 LPRINTTAB(30);"  X          XX    XX      X                ";
942 LPRINT"   X***X"
960 LPRINTTAB(30);" X           //XXXX        X                ";
962 LPRINT"   XXXX"
980 LPRINTTAB(30);"X           //    X                         ";
982 LPRINT" XX"
985 '----5----5---X#####X#####X#####X#####X#####X#####X#####X##X
1000 LPRINTTAB(30);"X          //     X          XXXXXXXXXXXXXX";
1002 LPRINT"XX/"
1020 LPRINTTAB(30);"X     XXX//      X          X"
1040 LPRINTTAB(30);"X    X   X       X         X"
1050 LPRINTTAB(30);"X    X    X      X        X"
1060 LPRINTTAB(30);"X    X    X      X        X                 ";
1062 LPRINT"   XX"
1070 LPRINTTAB(30);" X    X   X      X        X                 ";
1072 LPRINT"XXX  XX"
1080 LPRINTTAB(30);" X     XXX        X        X               X";
1082 LPRINT"  X X  X"
1090 LPRINTTAB(30);" X               X          X               X";
1092 LPRINT"X X  XXXX"
1100 LPRINTTAB(30);"  X               X         XXXXXXX\       X";
1102 LPRINT"X   XX  X"
1110 LPRINTTAB(30);"   XX             XX               X        X";
1112 LPRINT"    X  XX"
1120 LPRINTTAB(30);"     XX             XXXX   XXXXXX/        X ";
1122 LPRINT"    XXXX"
1125 '----5----5---X#####X#####X#####X#####X#####X#####X#####X##X
1130 LPRINTTAB(30);"       XXX              XXX***           X  ";
1132 LPRINT"  X"
1140 LPRINTTAB(30);"        XXXXXXXXXXXXXXX *   *         X   ";
1142 LPRINT" X"
1150 LPRINTTAB(30);"                         *---* X      X    ";
1152 LPRINT"X"
1160 LPRINTTAB(30);"                        *-* *   XXXX X    X"
1170 LPRINTTAB(30);"                        *- *       XXX   X"
1180 LPRINTTAB(30);"                       *- *X          XXX"
1190 LPRINTTAB(30);"                       *- *X  X          XXX"
1200 LPRINTTAB(30);"                      *- *X    X           ";
1202 LPRINT" XX"
1210 LPRINTTAB(30);"                      *- *XX    X           ";
1212 LPRINT"   X"
1220 LPRINTTAB(30);"                     *  *X* X    X          ";
1222 LPRINT"    X"
1230 LPRINTTAB(30);"                     *  *X * X    X         ";
1232 LPRINT"     X"
1234 '----5----5---X#####X#####X#####X#####X#####X#####X#####X##X
```

SNOOPY CALENDAR CONT. - - - - - -

```
1236 '----5----5---X#####X#####X#####X#####X#####X#####X#####X#X
1240 LPRINTTAB(30);"                    *  * X**  X   XXXX     ";
1242 LPRINT"     X"
1250 LPRINTTAB(30);"                    *  * X**  XX     X     ";
1252 LPRINT"     X"
1260 LPRINTTAB(30);"                   *  ** X** X     XX      ";
1262 LPRINT"    X"
1270 LPRINTTAB(30);"                   *  **  X*  XXX   X      ";
1272 LPRINT"   X"
1280 LPRINTTAB(30);"                  *  **    XX    XXX      X";
1282 LPRINT"XX"
1290 LPRINTTAB(30);"                 *  * *      XXXX      X   ";
1291 '----5----5---X#####X#####X#####X#####X#####X#####X#####X#X
1292 LPRINT" X"
1300 LPRINTTAB(30);"                *  * **          X     X   ";
1302 LPRINT" X"
1310 LPRINTTAB(30);" ==============  *  *           X     X   ";
1312 LPRINT"  XXXXXXXXX\"
1320 LPRINTTAB(30);" -      *        * *       /XXXXXX     XXX";
1322 LPRINT"XXXX\       )"
1330 LPRINTTAB(30);"    ==========* *      X               ";
1332 LPRINT"     )\     )"
1340 LPRINTTAB(30);"      ===*         *     X               \";
1342 LPRINT"   \ )XXXXXX"
1350 LPRINTTAB(30);"==========**********       XXXXXXXXXXXXXXX";
1352 LPRINT"XXXXX"
1355 '----5----5---X#####X#####X#####X#####X#####X#####X#####X#X
1360 LPRINT
1380 RETURN
1390 END
1500 ERA"SNOOPYCL"
1600 SAVE"SNOOPYCL":CLS:DIR
```

CHANGING SNOOPY CALENDAR FOR PRINTER/PLOTTER USE by Dave Boyce.

Now that You have typed in Part 2 of the Calendar we can procede. Load in the completed program and follow the instructions given below.

Delete Lines 162, 348, 558 and 820.

Type in -: SAVE"SNOCALPP" and press RETURN. (DISK SAVE)

Type in -: CSAVE"SNOOPY/CAL" and press RETURN. (TAPE SAVE)

Add Lines 70, 80, 160, 285, 595 and 1255 as Listed below.

Change Lines 90 to 120, 345, 365, 375, 385, 552, 580, 585, 590, 830 and 1500 as Listed below.

Also change the TAB(30)'s in Lines 850 to 1350 to TAB(15)'s as the example shows below.

Type in -: GOTO 1500 and press RETURN. (DISK SAVE)

Type in -: GOTO 1600 and press RETURN. (TAPE SAVE)

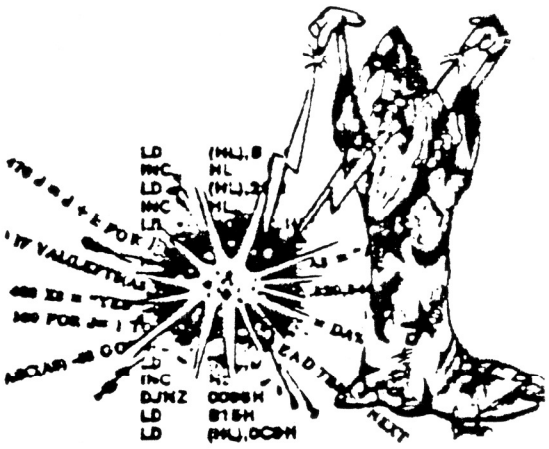
NOTE :- In case of DISK SAVE use line 1500 and line 1600 for TAPE SAVE whenever insructions call for program SAVE.

CONTINUED OVER PAGE

```
70 '  FILE - SNOCALPP
80 '  ##  SNOOPY CALENDAR ##
90 '  FOR THE PRINTER/PLOTTER ONLY
100 ' ## LINE 160 WILL SET YOUR PRINTER/PLOTTER
105 ' IN THE CORRECT MODE
110 ' ### RUN LINE 595 TO RETURN TO TEXT MODE
120 ' IF LEAVING PROGRAM PREMATURELY
160 LPRINTCHR$(18):LPRINT"S0":LPRINTCHR$(17)
285 LPRINT
345 LPRINT:LPRINT:LPRINT
365 LPRINTB$
375 LPRINT:LPRINT
385 LPRINT:LPRINTA$
552 GOSUB830:' SOOPY PRINTOUT
580 CLS:INPUT"ANOTHER YEAR ";YY$:IFYY$="Y"THEN RUN ELSE 595
585 END
590 ' ### RETURN TO TEXT MODE
595 LPRINTCHR$(18):LPRINT"S1":LPRINTCHR$(17)
830 ' SNOOPY PRINTOUT
845 '----5----5---X#####X#####X#####X#####X#####X#####X#####X##X
850 LPRINTTAB(15);"              XXXX"
860 LPRINTTAB(15);"             X    XX"
1255 '----5----5---X#####X#####X#####X#####X#####X#####X#####X#X
1400 :
1450 :
1500 ERA"SNOCALPP":SAVE"SNOCALPP":DIR:' THIS LINE FOR DISK ONLY
1550 :
1560 :
1600 CSAVE"SNOOPY/CAL":'TAPE USERS USE THIS LINE TO SAVE PROGRAM
```

ASSEMBLY

REPRINTED WITH PERMISSION MARK HARWOOD,FROM HIS VZ USER MAGAZINE



So you've played those fast video - games and wished you could write one yourself! You've played around with slow old BASIC, and just become more frustrated with graphiss that crawl along! Introducing Z80 Assembly Language-your programs will never be the same again! In this series, we'll be looking at Z80 Assembly language.

Let's start out by discussing the operation of the Z80 CPU. We need to understand that a microprocessor like the Z80 is only capable of carrying out a couple of basic functions. It can either read or write a value to some memory location, or carry out some operation (mostly mathematical) on a binary number it holds in one of its 'registers' (more about registers later). An assembly language program must carry out many of these processes in order to produce the desired operation. For this reason, Assembly language is known as a 'low level' language - its commands are very limitted when compared with a high level language like BASIC. However, because the language is so fundamental, and is the 'natural tongue' of the processor, the advantage in using it is high speed operation.

There is an important difference between Assembly Language and Machine Code, which should be pointed out at this point. A Machine code program consists only of numbers. The Z80 reads these numbers, and executes a specific function for a specific

value. Programming directly in machine code would require the programmer knowing the number which corresponds to every instruction! When programming in Assembly Language, a program, such as the Editor/Assembler, allows the programmer to use commands, known as Zilog Mnemonics, just as the BASIC programmer uses BASIC commands. These mnemonics consist of a few letters which have some meaning for a human being! For example, LD stands for 'load', CP for 'compare' etc. These commands are entered with line numbers, and when the program is complete, the Editor/Assembler is told to 'assemble' them into a machine code program. It does this by reading through the mnemonics and building up a chain of numbers which represent each instruction. This is known as the 'Object Code'. The original listing of mnemonics is the 'Source Code'. Although the Editor/Assembler offers some other handy functions, this is its prime function. Thus an Assembly Language program is made up of mnemonics, or 'op-codes', while a Machine Code program is simply the numeric equivalent of these instructions.

Before we can start programming, we need to know a little about what's inside the CPU. Although it consists of many thousands of components, making up many complex electronic circuits the programmer needs only to think of the CPU as a collection of 'registers'. A register is somewhat like a variable in BASIC, in that it stores a number (in binary form) on which mathematical operations can be carried out. They differ from BASIC variables in that they are limitted to holding either an eight-bit or sixteen-bit number, depending on the register. Strictly, each single register can hold only eight bits, though some registers are paired together to form a sixteen bit register pair. The registers of the Z80 are shown below:

Main Register (8 bits) Accumulator: A
Flag Register (8 bits) F
16 bit Registers: (made up of 2x8 bit registers)

	High byte	Low byte
BC Register Pair:	B	C
DE Register Pair:	D	E
HL Register Pair:	H	L
Index Register X:	IX	
Index Register Y:	IY	
Stack Pointer:	SP	
Program Counter:	PC	

The Accumulator, or A register, is the primary register. It is used for most logic operations and maths operations, though some of these functions are offered on the other register pairs. The F or Flag register is a special register used to indicate certain conditions and outcomes of calculations and logic operations. We'll be talking & using this in great amounts later.

Next comes the 'general purpose' registers. The B, C, D, E, H & L registers are all 8-bit, and can be accessed individually. However, they are also paired up as the BC, DE, and HL register pairs to form 16-bit registers. These are accessed as a whole. These are the registers which are used more generally by the programmer.

Of all the registers mentioned so far, there exists a 'duplicate' or alternate register, these making up an alternate register set. Only one set of registers can be accessed at a time, and the Assembly Language programmer need not be concerned with the alternate set as it is infrequently required.

The next two registers are known as index registers. They are used mainly when handling tables of data, and have special commands associated with them to allow such processing easy. We'll be looking at these in a later article.

The Stack Pointer SP is used by the Z80 to keep track of where the top of the stack is. The 'stack' is something which you need not be concerned with yet - but it is very important for storing data temporarily. The PC (Program Counter) is used by the Z80 to keep track of where abouts in memory the current instruction is.

Now that we've looked at the registers offered in the Z80, you're probably asking "But how can I use them?". This leads us into looking at our first Assembly Language command, or 'op-code' - loads. A load does simply what it says - data is loaded from one place into another. A number might be loaded from a memory location into a register, or from a register into a memory location. Alternatively, a register may simply be loaded with a number specified by the programmer. This type of load is called 'Immediate Addressing', and we'll be looking at it first.

The op-code mnemonic for a load is LD. Unlike BASIC, the setting out of Assembly Language instructions is important. For example, every LD must be followed by a space, then the 'operand'. The operand is that part of the instruction containing the articles to be handled in the instruction. For example, the instruction:

```
LD N,M
```

means 'load N with the data specified by M'. Let's take a real case. Suppose you wanted to load the A register with the number 255, using the Ed.Assem. you could do it in two ways: Either LD A,255 or LD A,FFH would load the Accumulator (A register) with the decimal 255. The second version simply uses the Hexadecimal notation (note the -H suffix). Remember that as the A register is 8-bit, 255 is the largest number it can hold. However, immediate addressing is also provided for the 16-bit register pairs: eg. LD HL,65535 will load the HL pair with 65535, the 'high' byte being stored in the H register, and the 'low' byte in the L register.

While we're on the subject of instruction formating let's take a look at what a complete line looks like with the Ed.Assem. To add lines to your program, or to start a new program, the 'I for insert' instruction is used. The Ed.Assem. will then put up a line number (the next in the program). Each line has four fields or 'areas'. These are the line number, the label, the op-code and the operand. Note that comment lines do not have this format (we'll be looking at these soon). The label is a four-lettered name given to the line, which is recognised by the Ed.Assem. In our example above, if we named the instruction 'LOAD', the line would look like this:

```
001 LOAD     LD HL,65535
```

No doubt, most of your lines won't have a label, so when entering such a line, hit the space bar before entering the op-code, or the Ed.Assem. will mistake it for a label!

Getting back to LDs, many other versions of the instruction are available. As we mentioned earlier, a load can involve moving data from a specific memory location to one of the registers. For example, suppose we wanted to load the contents of memory location 6800H into the A reg. - we would use:

```
LD A,(6800H)
```

Note that this is very similar to the immediate mode, except that the operand is enclosed in parenthesis. This means that instead of loading A with 6800H (which is impossible!) A is loaded with the contents of MEMORY LOCATION 6800H. In Z80 Assembly, parenthesis always indicate that the operand enclosed is being used to point to a memory location. This is a very important concept. The format above can be reversed, so that a memory location is loaded with the contents of a register, such as:

```
LD (6800H),A
```

This loads memory location 6800H with the contents of A. Note that when 16-bit registers are used in this op-code, two bytes of memory will be involved. LD HL,(6800H) will load the low byte from 6800H into L and the high byte from 6801H into H. Similarly, LD (6800H),HL loads 6800H with L and 6801H with H. This is known as Absolute Addressing.

Another very useful version of this mode is Indirect Addressing. Here, the address involved in the load is pointed to by one of the 16-bit register pairs. For example, we could have achieved the same result as above using:

```
LD HL,6800H
LD (HL),A
```

The first line loads HL with the number 6800H. In the next line, the ADDRESS POINTED TO BY HL is loaded with the contents of A. Thus HL, having been loaded directly with 6800H points to this address in memory in line 2. This method has its advantages when the memory location is constantly changed by changing the contents of the register pointing to it.

Although we've covered most of the forms of loads offered by the Z80 here, there are obviously different versions of each to allow for all the registers. In order that you are aware of which LDs are allowed, we recommend that you obtain a listing of all the Z80 op-codes. These are printed on charts, often hard covered, and are available from some electronics outlets. They provide a complete listing of the Z80 Zilog instruction set, and will let you get the most out of your Assembly Language programs. We'll now consider a couple of other simple instructions:

NOP : This is the 'no operation' instruction - the Cpu simply moves on to the next instruction after a delay of 4 clock cycles. This instruction is generally used only as a time delay or to allow 'overlaying' of other routines later.

Next month we return with a look at many more commands and our first simple program.

UNDERSTANDING YOUR VZ - - PT.5

By R.QUINN

POS will give you the current location of the cursor in whatever pair of display lines it has moved to. This is useful for keeping track of the cursor while a program is running since the cursor is invisible under program RUN. However, POS is not always reliable, and since it returns values in the range 0 to 63 only, it doesn't tell you what pair of display lines the cursor is in. Fortunately there is a pointer in the communications region which keeps track of the cursor in terms of its absolute address in text mode video memory (addresses 28672 to 29183). The pointer to cursor position occupies addresses 30752 and 30753, and can be accessed thus: X=PEEK(30752)+PEEK(30753)*256

You can use the value of X (or whatever numeric variable you choose to use) in your programs to control the display with absolute certainty as to where you're at. Of course, if you PRINT X then the cursor will no longer be at the address X, will it? X = X - 28672 will then return the cursor position in PRINT@ format where screen addresses have a range of 0 to 511.

STRING$ allows you to create and display a string of umpteen identical characters. The STRING$ format is STRING$(B,C) where B is a number in the range 0 to 255, or a numeric variable, which determines the number of characters in the string. C can be a number or a numeric variable with a range 0 to 255, or it can be a string or a string variable. As a number, C is interpreted as a VZ ASCII character code, converting to CHR$(C). Examples: X$=STRING$(32,95):PRINT STRING$(255,140) 'This displays a string of 255 identical graphic characters. You will need to CLEAR enough memory for string space to accomodate long strings like this. If C as a string or string variable has more than one character, STRING$ will use the first character of the string.

PRINTSTRING$(X,A$) STRING$ can also be used with PRINT@

You can experiment with the STRING$ statement with this one line program:

```
10 INPUTB:INPUTC:FORR=1TO15:PRINTLET(B,C):NEXT:GOTO10
Now enter these commands:   CLEAR999:POKE31484,196:RUN
```

At, and marking, the end of a basic program in program memory are two zero bytes. The last line in a program points (with its start address of next line pointer) to the first of these zero bytes as the start of the next line. But this 'next' line is a fake. Only these first two bytes of this pseudo line exist and they point to address zero. So you will always find three zero bytes at the end of a program--the zero byte marking end of last line, then the two zero bytes starting the pseudo last line. The next byte is the beginning of the VARIABLE LIST TABLE and it is the address of this byte that is pointed to by the END OF PROGRAM pointer, which is really a START OF VARIABLE LIST TABLE pointer. The VARIABLE list table lists all the variables--numeric and string--that are used when a program runs. Simple numeric variables and pointers to simple string variables are entered into the simple variables table on a first come first served basis. Array variables are likewise dealt with in the array variables table, which always starts at the end of the simple variables table. The first byte of each entry is a code which identifies the variable type: '2' is integer; '3' is string; '4' is single precision; '8' is double precision. The next two bytes code for the variable name.

CONTINUED OVER PAGE

VARPTR is one of the most useful of the basic words that were left out of the VZ's vocabulary. It can be used to explore the way in which variables are located, organised and handled in memory.

PRINTVARPTR (variable) will display the address of the fourth byte of the entry for the variable in the variable list table. Only simple string variables will be considered further here.

Z = VARPTR(A$) Z is the address where the length of A$ is stored.

X = PEEK(Z) Then X is the length of A$.
The bytes with addresses Z + 1 and Z + 2 are then a pointer in the variable list table to the start address of the content of A$.

Y = PEEK(Z+1) + PEEK(Z+2) * 256 then Y is the start address of the content of A$.

With this information we can change the length of a string and the start address of a string's content.

```
POKE X,A  or POKEVARPTR(A$),A
POKEZ+1,B  or POKEVARPTR(A$)+1,B
POKEZ+2,C  or POKEVARPTR(A$)+2,C
```

New your VZ and enter line 20:

```
20 POKELET(X$)+1,0: POKELET(X$)+2,112
```

Now POKE31470,192:POKE31481,192 and enter line 10:

```
10 X$=LET(32,32)
```

Now POKE31472,196:CLS:RUN
Invoke inverse (CTRL :), then position cursor on top screen line and type in whatever characters you please. Bring the cursor down the screen and PRINTX$. What we have done is given X$ a length of 32 and pointed it to the top display line of video screen so that the content of X$ will be whatever characters occupy the first 32 cells of video memory. Inverse mode entry is required for alphabetical characters and a few other characters because of the difference between ASCII character codes and VZ PEEK/POKE character codes. I leave you to work it out and to explore the possibilities of pointing strings to any part of memory (ROM or RAM) with any length from 1 to 255 bytes. Design your own INPUT routines!

Some string pointers point to locations in program memory (A$ = "LITERAL STRINGS"); others point to locations in string space (B$ = "NON" + A$). Your VZ will reserve 50 cells at the top of user memory for string space. If you want to reserve more (or less) string space memory then you use the CLEAR n command where n is the amount you want.

Note :- Next issue will contain concluding chapter on UNDERSTANDING YOUR VZ by ROBERT QUINN.

© COPYRIGHT R.QUINN

SOFTWARE FOR SALE FROM VZ DOWN UNDER

Prices shown include postage and packing anywhere in Australia. New Zealand customers, please add A$2.00 to the prices below. Make sure that the memory of your VZ system is greater than or equal to the memory required by the software that you wish to purchase. Memory sizes indicated do not include Video Ram. Prices specified are calculated as tape based software. Disk copies are available for $2.00 extra. All prices are in Australian dollars.

CASTLE GREYSTONE	32K	$12.50	SNAKE DUEL (JSTKS)	16K	$6.50
DATA	16K	$20.00	BLACKJACK ROYALE	16K	$12.50
GALACTIC EMPIRES	16K	$12.50	STELLAR WARS	16K	$6.50
HAUNTED MANSION	22K	$12.50	TRIVIAL CULT	16K	$6.50
KNIGHTS QUEST	32K	$15.00	VGRAPHIXZ	16K	$15.00
NAMESTAR/SHOPSTAR	22K	$12.50	VZ-MONOPOLY V2.0	22K	$12.50
SCOTLAND YARD	16K	$6.50	VZ-MONOPOLY V2.0	32K	$15.00
GRAPHSTAR	22K	$20.00			

ADVENTURER'S COLLECTION - Contains SCOTLAND YARD, HAUNTED MANSION, KNIGHTS QUEST & CASTLE GREYSTONE. All on the one cassette tape - $36.50

ARCADE COLLECTION - Contains GALACTIC EMPIRES, STELLAR WARS, SNAKE DUEL & your choice of 32K or 22K VZ-MONOPOLY. All on the one cassette tape - $30.00

Please make all cheques and money orders payable to :-

Scott Le Brun
59 Brentwood Dve
Wantirna VIC 3152

P.O.BOX 1972 C.P.O AUCKLAND N.Z

(C) AUCKLAND VZ 300/200 COMPUTER USER'S CLUB 1986

ALL MATERIAL IN THIS NEWSLETTER IS SUBJECT TO COPYRIGHT..
CONTRIBUTED PROGRAMS & ARTICLES CAN BE REPRODUCED ON THE UNDERSTANDING THAT THEY ARE FOR THE PRIVATE USE OF SUBSCRIBERS ONLY.
COPYRIGHT IS ALWAY'S RETAINED BY THE AUTHOR.
THIS CLUB WILL NOT BE HELD RESPONSIBLE FOR THE BREECHING OF COPYRIGHT CAUSED BY THE PUBLICATION OF ANY MATERIAL SENT IN UNDER A FALSE NAME.

YOU ARE WELCOME TO PHONE ME BETWEEN 6.30 PM AND 9 PM ----PHONE 673-967

MTS PRESENTS

VZ Screen Edd is a new powerful utility program for the VZ computers. The utility allows easy and fast drawing of high resolution screens for use in what ever you please.

The program is fitted out with a whole host of features including instant backgrounds, built in lettering, many types of printer dumps, a fully transperant cursor, fast screen movement and the option of using joystick. The utility is also fitted out with both disk and tape functions and is standard for any set up as it makes system checks before trying anything that may not be connected such as a disk drive.

The utility's most useful feature though is its ability to both load and save to tape. No screen editor has previously been able to do so in a sucessful and usable way. Here begins the world of screen shows for tape owners. Yes, as described in the manual it is possible to make a screen show on tape.

The utility comes complete with a six page manual which also contains a video worksheet for your own use and on top of that there is also four example screens to get you started. The manual fully explains how to use the utility, how to use your designed screens and how to go about designing a screen from start to finish. A full list of commands is also in the manual. The utility also, because of its setup, is able copy screens easily between tape and disk and between tape and tape etc.

Overall this utility presents a very worth while investment for both tape and disk owners. Disk owners can also use the tape facility as a reliable back up for their screens. Who knows what someone will dream up as a usage for it.

PRICE $25.00 + $2.00 P&H

SEND YOUR CHEQUE OR MONEY ORDER TO :-

Matthew Taylor
38 Fishing Point Road,
RATHMINES NSW 2283 AUSTRALIA

(049) 752350

MTS
MATTHEW TAYLOR SOFTWARE